LLNL-TR-652476

LAWRENCE LIVERMORE NATIONAL LABORATORY

A High Performance Computing Scheduling and Resource Management Primer

D. H. Ahn, J. E. Garlick, M. A. Grondona, D. A. Lipari, R. R. Springmeyer

March 31, 2014

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

A High Performance Computing Scheduling and Resource Management Primer

Dong Ahn, Jim Garlick, Mark Grondona, Don Lipari, and Becky Springmeyer

This primer introduces the concepts of High Performance Computing, scheduling, and resource management. It focuses first on the state of the art and then introduces the issues under consideration in the design of a next generation resource manager known as Flux. The Flux design addresses not just the challenges of resource management at a much larger scale of systems, but it also enables and promotes a much broader and richer model that invites collaboration with areas whose requirements cannot be met in current systems.

The term High performance computing (HPC) as it is commonly used refers to a category of computing systems that combine computing power from multiple units to deliver vastly higher performance than desktop computers can provide. HPC is distinguished not just by computing power. HPC systems are designed to run a variety of large applications for extended periods of time in order to solve complex problems. HPC is distinguished from web servers in that they dedicate requested resources to run applications provided by users. Current HPC systems are distinguished from cloud services like AWS in that they generally provide a single operating system with a runtime environment managed by the facility.

An HPC cluster is distinguished from a group of networked computers in that a cluster has (mostly) uniform hardware, installed software, and configuration. An HPC cluster typically has a high speed switch to enhance node to node communication. HPC applications involve multiple processes (tasks) running on multiple nodes in constant communication. MPI is the library of choice for task to task communication across node boundaries.

HPC scheduling is also separate from data-driven scheduling like Hadoop. The main difference is that data designs rely on the data residing on the compute clusters for extended periods of time. Jobs come and go to search and process that data, but by and large, the data is stays resident. HPC clusters have to provide a clean slate for every job. Exceptions exist, but for the most part, data needs to be retrieved at the start of the job and saved at the end of the job, typically to an external parallel file system.

HPC clusters are also distinguished by a batch scheduler and resource manager. Users submit “jobs” to the batch system and the scheduler decides when and on what resources to launch the job. The resource manager is the entity responsible for launching the job and executing the codes the user specifies across the allocated resources.

The most popular commercial batch schedulers are IBM’s Platform LSF, Adaptive Computing’s Moab Workload Manager, Altair’s PBS Professional, and Univa’s Grid Engine (formerly Sun Grid Engine). Popular open source batch schedulers are Maui, Slurm and Globus.

The most popular resource managers are Slurm, PBS, and TORQUE. Slurm does both batch scheduling and resource management.

A similar, but different means for utilizing large clusters is High Throughput Computing (HTC), where large numbers of relatively short lived, smaller sized jobs are processed. The Condor batch scheduler is tuned for HTC.

In both of the above, the usage pattern is the same. Users submit jobs to the scheduler and they eventually run on the computing resources the scheduler allocates. The most common thing submitted to the batch system is a job script. The job script typically starts with embedded directives that describe the set of resources on which it wants to run. It is then followed by one or more execution invocations of the user's code. An example script:

```
#!/bin/bash
#SBATCH –N 8       # give me 8 nodes
#SBATCH -t 5       # run me for 5 minutes maximum
#SBATCH -p pdebug # run me on nodes from the pdebug pool

srun –n 64 –i myInput –o myOutput myExe
```

Each batch system has a well-defined set of commands for interacting with the batch system. There are commands to submit a job, modify a job, signal a job, and cancel a job. In addition, there are commands that display the set of available resources and the queue of jobs, both running and waiting to run.

Modern batch schedulers provide the user with a clearer picture of when their jobs are projected to run, why their jobs may not be being scheduled, a detailed picture of current resource utilization, and historical records of completed jobs and resource usage.

Batch systems were originally designed to service a single cluster. Users submitted jobs to the cluster and once the job ran, users would examine the output. Larger centers developed or procured batch schedulers that would combine multiple clusters into a single system (aka grid). Users could then submit jobs to a single scheduler and the scheduler would decide on which cluster and which nodes to run the job. Status commands would retrieve the queue of jobs from all the clusters in the grid.

User applications typically need to be profiled and debugged, particularly in their development stages. HPC clusters traditionally offer a set of tools that can attach to and monitor running applications and allow them to be single stepped. Examples are TotalView and STAT.

Scheduling and Resource Management

Most batch schedulers follow a similar pattern. They prioritize the queue of pending jobs, and then attempt to run the highest priority job. If there are no resources yet available to service the top priority job, the scheduler's behavior can take two paths. It can either wait until enough running jobs complete and resources become available to run the top priority job. Or it can decide to look at the next priority job (or jobs) and run them on the resources that are available.

Schedulers have two main operations that can be tailored. The first is the job prioritization formula. The second is the method for selecting the resources to allocate to the top priority job.

The configuration of each of the above operations makes up what is known as the center's policy. The computing center management effectively enters into an agreement with their users to establish and then enforce a well-defined policy for how jobs are selected to run and what resources their jobs are allocated. Users need to know that they will receive the service they are entitled to and evidence needs to be available to convince users that they are received the service they were promised or sold.

Fundamentally, the scheduling problem reduces to how the jobs are prioritized and which (and when) resources are allocated to the job.

Factors that influence these decisions start with the computing resources available along with any constraints on their use. A compute node may have a limit on the number of jobs that can run on it simultaneously as well as an access list of users who are authorized to run jobs on it.

The next set of factors comes from the job request itself. Jobs typically request a number of nodes or processors on which to run, some memory perhaps, and the length of time they will need to complete. But in addition, a job can specify many other constraints: the pool of resources (aka queue), the need to receive mail at various state changes, the file systems to access, dependencies on other jobs, etc.

The complete list of specs can been seen by examining the man page for the batch submission commands, e.g., `sbatch` for Slurm, `msub` for Moab.

To recap, the scheduler implements policy to decide how to prioritize all the jobs submitted to the batch system. Then the scheduler has to consider all of the resources under its control, the constraints on those resources, and the job specification request to find the soonest time and most appropriate set of resources on which to run the job.

Current Scheduler Abilities – Job Prioritization

The scheduler can consider a number of factors in calculating job priority. Each of the following factors is typically normalized to 0.0 to 1.0 and configurable weights are applied:

- queue wait time – here an age (e.g., 2 weeks) must be defined as the limit (i.e., 1.0)
- job size – the higher the number of resources the job requests, the higher (or optionally lower) this value becomes.
- fair-share
    - under-serviced numbers range from 0.5 to 1.0 (most underserviced)
    - over-serviced numbers range from 0.5 to 0.0 (most over serviced)
- quality of service (QoS)

Current Scheduler Abilities – Resource Selection

The majority of resource requests to current HPC schedulers are for either nodes or cores. Required memory could also be included in the request, as can items such as disk space or GPUs. However, most schedulers primarily look for a number of nodes or cores to allocate to the job for a prescribed time period.

As simple as this begins, more sophisticated schedulers look to provide a collection of resources that satisfy specific constraints. They may be asked to allocate only nodes on a particular switch to minimize node-to-node communication delays. Schedulers may reserve higher memory nodes for jobs that really need them. Some applications require a specific system architecture and the scheduler must supply only those nodes that fulfill that request. Some schedulers look to pack jobs into a small set of contiguous nodes, while other schedulers are tailored to spread the jobs across as many nodes as possible to distribute the workload. In a similar vein, some schedulers look to pack jobs onto a small set or nodes so that banks of idle nodes can be put into a low power state or powered off.

In general, the degree of flexibility for scheduling jobs to resources is complex as each scheduling scenario is supported or enforced.

Current Scheduler Abilities – Job and Resource Accounting

An HPC batch system must generate and provide a complete set of statistics for the jobs that are run and the resources that are consumed. This begins with a list of job records, both individual and aggregated. Users and managers will want to look back over time to see what jobs ran, what resources they ran on, and whether they ran successfully to completion. Managers also require reports of how computing resource investments are utilized and by which users, groups and projects. Fair-share calculations are based off the difference between what resources were promised and which resources were ultimately used.

Accounting data is typically archived into databases that can be searched on demand. Periodic reports of job usage and resource utilization require these databases to provide information of interest to users, administrators, managers and funders.

Flux

The need for Flux grew out of the growing number and size of our computing resources. Existing batch schedulers bog down at the number and size of clusters we currently site. In addition, existing schedulers were not designed to schedule ancillary yet vital elements such as network traffic, power consumption, and licenses.

Existing schedulers do not support a dynamically changing resource allocation. Jobs that need to grow or shrink in size as well as migrating new resources to replace failing components are paradigms that are beyond most current schedulers.

The Flux Scheduler

Flux will offer a novel solution to the congestion experienced by a centralized scheduler that schedules all the resources on all the clusters in the grid. The Flux design calls for a hierarchy of jobs within jobs with each job allowed to install its own scheduler with as much or as little intelligence as it needs.

Our vision calls for the ability to create a varied collection of schedulers. We expect to offer a set of solutions, from simple to complex, along with the ability for users to create their own, given the framework that we provide. So, as we discuss schedulers, we are really discussing a range of potential solutions that can ultimately address the problems we identified in the Flux vision doc.

One can imagine then a simple scheduler that runs jobs in a first in, first out order. Additionally, this scheduler could select the first resources it finds that match the job specification. We will probably create just such a scheduler and supply it by default for users who only need FIFO scheduling.

At the other end, we plan to develop a collection of more sophisticated schedulers that handle the full spectrum of job requests from a disparate set of users. These are the schedulers that will need to deliver the center's service level agreements.

The primary function of the scheduler will be to identify computing resources that match the job specification. The resources will be described in a well-defined way. Resources have traditionally been defined in a flat file which is read by the scheduler on startup or whenever the file contents changes.

The Flux resource specifications need to be rich enough to support the requirements for all of the schedulers that can ever be developed, however simple or complex they may be. We expect the job specification will resemble the way the resources are described. This will help to minimize the effort to match requested resources to available resources.

Flux schedulers will not be limited to the traditional paradigm of scheduling the top priority job. Instead, the Flux scheduler framework will provide the freedom to innovate. Some of the novel approaches to scheduling include:

- Supporting MPI Malleable Applications upon the OAR Resource Manager

  http://www.lbd.dcc.ufmg.br/colecoes/colibri/2009/036.pdf

- Omega: flexible, scalable schedulers for large compute clusters

  http://eurosys2013.tudos.org/wp-content/uploads/2013/paper/Schwarzkopf.pdf

- Matching Techniques for Resource Discovery in Distributed Systems Using Heterogeneous Ontology Descriptions

  http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.8073&rep=rep1&type=pdf

- Ontology Based Service for Grid Resources Description

  http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1558548

- A Constraint Language Approach to Grid Resource Selection

  http://www.cs.uchicago.edu/files/tr_authentic/TR-2003-07.pdf

- Efficient Resource Description and High Quality Selection for Virtual Grids http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1558608

- Moldable Job Scheduling for HPC as a Service with Application Speedup Model and Execution Time Information

  http://www.ftrai.org/joc/vol4no4/5-A.pdf

- An RMS for Non-predictably Evolving Applications

  http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6061151

The frontiers for scheduling that Flux will pioneer include:

- Job elasticity – scheduling resources that change during the life of the job.
- Job resiliency – scheduling resources to replace overly loaded or failing resources.
- Power awareness – scheduling resources that conform to time varying power utilization constraints.
- Resource variety – scheduling a rich set of resources both on and off the cluster that go beyond the traditional cores, memory, and nodes.
- Network topology and I/O – selecting resources that conform to communication bandwidth and delay constraints.